DIGITÓPOLIS



Jose David Cuartas Correa

DISEÑO DE APLICACIONES INTERACTIVAS PARA CREATIVOS Y COMUNICADORES





Institución Universitaria
Los Libertadores

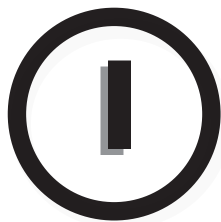
Jose David Cuartas Correa es Diseñador Visual y estudiante del Doctorado en Diseño y Creación, en la Universidad de Caldas (Manizales, Colombia). Profesor de tiempo completo y coordinador del área de investigación del programa de Diseño Gráfico, en la Fundación Universitaria Los Libertadores (Sede Bogotá) y profesor catedrático del programa de Diseño Industrial, de la Universidad Jorge Tadeo Lozano.

El profesor Cuartas es promotor del uso y desarrollo de software libre, de la cultura libre y de las tecnologías emergentes en el ámbito del arte, el diseño y el entretenimiento. Es usuario de Software Libre desde el año 1999 con experiencia amplia en programación multimedial y creación de instalaciones interactivas.

Entre las principales áreas de investigación, en las que trabaja, se encuentran las interfaces tangibles y perceptuales de usuario; específicamente, cuenta con varios desarrollos en torno a la tecnología de Realidad Aumentada, como el software ATOMIC Web Authoring Tool y el software ATOMIC Authoring Tool.

En el año 2009 fue estudiante de intercambio en Advanced Diploma in Experimental Media Arts en CEMA (Center of Experimental Media Art) de la Universidad de Srishti School of Art, Design and Technology, en la ciudad de Bangalore, India, en donde además dictó un curso de "Diseño de Interface e Interacción en la Web".

DIGITÓPOLIS



Jose David Cuartas Correa

DISEÑO DE APLICACIONES INTERACTIVAS
PARA CREATIVOS Y COMUNICADORES





Institución Universitaria
Los Libertadores

Cuartas Correa, Jose David. Digitópolis I: Diseño de aplicaciones interactivas para creativos y comunicadores / José David Cuartas Correa. -- Bogotá: Fundación Universitaria Los Libertadores. Producción Editorial , 2014. 110 p.
ISBN: 978-958-9146-46-0
1. TECNOLOGÍAS DE INFORMACIÓN Y LA COMUNICACIÓN. 2. INNOVACIONES EDUCATIVAS 3. TECNOLOGÍA EDUCATIVA. I. Cuartas Correa, Jose David II. Tít. III. Fundación Universitaria Los Libertadores

303.4833 / C961d

PRIMERA EDICIÓN: ABRIL, 2014

LOS LIBERTADORES, FUNDACIÓN UNIVERSITARIA
Bogotá D. C., Colombia
Cra 16 No. 63 A - 68 / Tel. 2 54 47 50
www.ulibertadores.edu.co

Pedro Bellón
Director Centro de Producción Editorial

Francisco Buitrago Castillo
Revisión Editorial

María Fernanda Avella Castillo
Diagramación Portada

Jose David Cuartas Correa
Diseño y Diagramación

Editorial Kimpres Ltda.
Impresión

PARTE UNO:

LÓGICA BÁSICA DE PROGRAMACIÓN

1.1. ¿Qué es programar?

Programar es como cocinar. Lo que se hace es seguir una receta, en una serie de pasos ordenados que indican qué se debe hacer y en qué momento. La programación se basa en darle solución a diferentes tipos de problemas; y pueden existir múltiples soluciones para un mismo problema. Al programar, lo primero que se debe determinar son las entradas, los procesos y las salidas, las cuales se definen respondiendo a las siguientes preguntas:

Entradas:	¿Cuáles serán los datos que deben ingresarse al programa?
Procesos:	¿Cómo serán procesados los datos?
Salidas:	¿Cuáles serán los resultados que le serán presentados al usuario?

1.2. ¿Cómo bocetar programas?

Antes de entrar a programar en el lenguaje Processing, se debe bocetar la solución del problema. El Pseudocódigo y los Diagramas de flujo son dos de las principales estrategias para esto.

- **Pseudocódigo**

El Pseudocódigo es un texto, en el que cada línea contiene cada uno de los pasos que se deben seguir para solucionar el problema que se desea programar. Si, por ejemplo, se quisiera hacer el Pseudocódigo de los pasos que se deben seguir para hacer una llamada telefónica, se podría pensar que consistiría en:

1. Levantar el teléfono
2. Marcar
3. Hablar
4. Colgar

Pero un buen pseudocódigo consideraría situaciones como: ¿Qué pasa si el teléfono está ocupado?, o ¿hay tono al levantar el teléfono? Ejemplo:

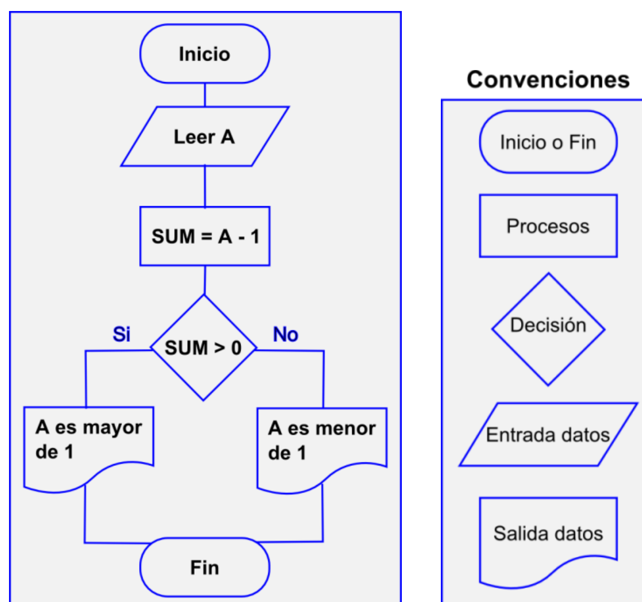
1. Levantar el teléfono
2. Verificar si hay tono
3. Si no hay tono, colgar e ir al paso 1
4. Si hay tono, entonces marcar
5. Si está ocupado, colgar e ir al paso 1
6. Si contestan, entonces hablar
7. Si se termina de hablar, entonces colgar

- **Diagramas de flujo**

Los Diagramas de flujo son una forma gráfica de visualizar lo que pasa en un programa. El riesgo que se corre con los diagramas de flujo es que a veces puede resultar difícil codificar el diagrama. Sin embargo, son

una herramienta de gran ayuda para visualizar, de forma rápida, los puntos de decisión durante la ejecución del programa que se está diseñando.

Un ejemplo de diagrama de flujo podría darse cuando se pida al usuario que introduzca un valor A, luego que a ese valor se le reste 1 y se determine si el valor A es mayor o no que 1. Sería algo así:



1.3. Datos en memoria

Todos los programas hacen uso de la memoria para acceder a los datos que se van almacenando, mientras el programa se está ejecutando. Los programas separan espacios en la memoria, para almacenar

diferentes tipos de datos, los cuales pueden ser: enteros, decimales, caracteres, cadenas de caracteres, valores binarios, valores hexadecimales, etc.

Cada lenguaje de programación define con cuales tipos de datos trabajar. Para el caso particular de este texto, se usarán solo algunos de los tipos de datos disponibles en Processing, como lo son: **int** (entero), **float** (decimal), **char** (caracter), **String** (cadena de caracteres), y **boolean** (booleano).

Para poder trabajar con estos datos, es necesario conocer las diferentes formas que existen para reservar dichos espacios de memoria. Principalmente, se habla de variables, constantes y arreglos.

- **¿Que son las variables?**

Son espacios reservados en la memoria, para almacenar un dato, que puede ser numérico, alfanumérico o alfabético. Un ejemplo podría ser la hora exacta, puesto que es un valor que, para considerarse correcto, debe actualizarse a cada instante. Es decir, no puede ser estático, sino estar cambiando a todo momento.

- **Tipos de variables**

Las variables pueden ser de dos tipos: locales o globales.

Variables locales: Son las que se definen dentro de una función. Se crean al entrar a la función y se destruyen al salir. Este tipo de variables también pueden estar definidas dentro de un bloque de código (Conjunto de línea de código que estén entre llaves) y ser locales respecto a ese bloque, creándose al entrar en él y destruyéndose cuando salen.

Variables globales: Están definidas fuera de las funciones y pueden ser usadas por todas las funciones.

- **¿Cómo se crea una variable?**

Para crear o declarar una variable se pone el tipo de dato, seguido del nombre de la variable y se agrega un punto y coma al final, así:

```
TipoDato NombreVariable;
```

Un ejemplo sería declarar la variable de nombre "letra" de tipo caracter, así:

```
char letra;
```

- **¿Cómo declarar variables de un mismo tipo?**

Para declarar distintas variables de un mismo tipo, se pone la clase de dato, seguida de los nombres de las variables, separados por comas así:

```
TipoDato Var1, Var2, Var3,...., VarN;
```

Un ejemplo sería declarar de tipo entero, las variables "hora", "minuto" y "segundo", así:

```
int hora, minuto, segundo;
```

- **¿Qué es una constante y cómo se declara?**

Es un valor que no cambia dentro de toda la ejecución del programa. Dicho valor puede ser numérico, alfanumérico o alfabético. Para definir una constante, se usa la siguiente estructura:

```
final TipoDato NombreConstante = Valor;
```

Un ejemplo sería declarar la constante numérica de nombre "PI", con el valor decimal 3.14159

```
final float pi = 3.14159;
```

- **¿Que son los arreglos?**

Un arreglo es un espacio reservado en la memoria, para almacenar un conjunto de datos del mismo tipo, los cuales están dispuestos de forma consecutiva.

Para acceder a cada uno de los datos almacenados en el arreglo, hay que usar el nombre del arreglo y su subíndice. Los subíndices indican la posición dentro de los arreglos y dichas posiciones se cuentan a partir de cero. Así pues, la primera posición de un arreglo **X**

seria X[0] y se leería: "**Equis sub cero**". Los subíndices de un arreglo pueden ser una variable, una constante o una operación matemática entera.

Los arreglos pueden ser unidimensionales, bidimensionales, tridimensionales o multidimensionales. Popularmente, los arreglos unidimensionales son conocidos como vectores (si almacenan datos numéricos) o como cadenas (si almacenan datos alfabéticos), mientras que los arreglos bidimensionales son conocidos cómo matrices.

- **Vectores**

Los vectores, también conocidos como listas, son arreglos unidimensionales que cuentan con un solo subíndice, el cual indica el tamaño de vector, es decir, el número de posiciones contiguas del mismo tipo de dato. Su estructura básica es la siguiente:

```
Vector [Tamaño];
```

Un ejemplo de cómo se reserva el espacio en memoria de un vector con nombre A, y de un tamaño de seis posiciones, se vería así:



En este ejemplo, la posición A[0] tendría asignado el valor 5, mientras que la posición A[5] tendría asignado el valor 8. En Processing, los vectores se declaran bajo la siguiente estructura:

```
TipoDato[] Nombrevector = new TipoDato[Tamaño];
```

Se pone primero el Tipo de Dato (int, char, float…etc), seguido del subíndice vacío ([ ]). Después se pone el Nombre del vector, seguido por un igual y, por último, se pone la palabra "new" y el Tipo de Dato seguido del subíndice con el número de posiciones a reservar.Un ejemplo sería declarar un vector que almacene la edad de 10 personas, así:

```
int[] edades = new int[10];
```

Si se quisiera asignar la edad de la primera persona del vector, se haría de esta manera:

```
edades [0]= 21;
```

- **Cadenas**

Una cadena es un tipo de arreglo unidimensional, que almacena valores de tipo caracter. En Processing, las cadenas se pueden declarar de dos formas. Como un Arreglo de caracteres, así:

```
char[] NombreArregloCaracteres = new char[tamaño];
```

O como una clase de tipo String, así:

```
String NombreCadena;
```

La diferencia entre las dos formas es que, cuando se trata de mostrar en pantalla un arreglo de caracteres, se debe incluir carácter por carácter, En cambio, usando la clase String, se puede mostrar la cadena de caracteres, toda seguida, de forma rápida y sencilla. Para usar un arreglo de caracteres como si fuera una cadena, se debe entonces validar de la siguiente forma:

```
New String(NombreArregloCaracteres);
```

También es posible crear vectores de cadenas. Un ejemplo podría ser declarar un vector llamado "direcciones", que almacene los nombres de los cuatro puntos cardinales, y que después muestre dichos nombres en la ventana de depuración, así:

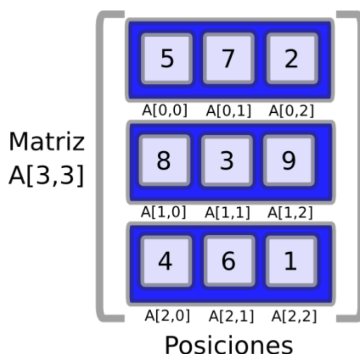
```
String[] direcciones = new String[4];
direcciones [0] = "Norte";
direcciones [1] = "Sur";
direcciones [2] = "Este";
direcciones [3] = "Oeste";
println(direcciones [0]);
println(direcciones [1]);
println(direcciones [2]);
println(direcciones [3]);
```

- **Matrices**

Las matrices, también conocidas como tablas, son arreglos bidimensionales que cuentan con dos subíndices, el primero indica el número de columnas y el segundo, el número de filas. Su estructura básica es la siguiente:

```
Matriz [NúmeroColumnas][ NúmeroFilas];
```

Un ejemplo de cómo se reserva el espacio en memoria de matriz, con nombre A, de un tamaño de tres (3) columnas y de tres (3) filas, se vería así:



En este ejemplo la posición A[0][0] tendría asignado el valor 5, mientras que la posición A[2][1] tendría asignado el valor 6. En Processing, las matrices se declaran bajo la siguiente estructura:

```
TipoDato[ ][ ] NombreMatriz = new TipoDato[Columnas] [Filas];
```

Se pone primero el Tipo de Dato (int, char, float…etc), seguido de los dos subíndices vacios ([ ][ ]), después se pone el Nombre de la Matriz, seguido de un igual y por último se pone la palabra "new" y el Tipo de Dato seguido de los subíndices con el número de posiciones a reservar.

En el primer subíndice se pone el número de columnas que se desea crear y en el segundo subíndice se pone el número de filas que se desea crear

Un ejemplo podría ser crear una Matriz para almacenar las edades y las cédulas de 10 personas diferentes.

```
int[][] edadced = new int[10][2];
```

Si se quisiera asignar los datos de la primera persona de la matriz y poner en la primera columna las edades y en la segunda las cedulas, se haría así:

```
edadced [0][0]= 18;
edadced [0][1]= 75000000;
```

Así pues, el primer subíndice ([]) se usaría para seleccionar la persona y el segundo, para seleccionar el dato; la posición 0 para la edad; y la posición 1 para la cédula.

1.4. ¿Cómo procesar datos en memoria?

Para el acceso y procesamiento de datos en memoria, es indispensable hacer uso de unos caracteres especiales llamados "operadores". Los operadores pueden ser:

- De asignación
- De incremento y decremento unitario
- Relaciónales
- Lógicos
- Aritméticos
- Condicionales

- **Operador de asignación**

Asignación simple: Cuando con el operador de asignación le damos un valor a alguna estructura de almacenamiento de datos, ya sea una variable, un vector, una cadena, una matriz, etc. Su estructura básica es la siguiente:

```
Var = Expresión;
```

Donde Expresión puede ser un valor fijo o el de otra variable, una operación o un conjunto de operaciones, o puede ser una formula.

Asignación compuesta: Cuando el operador de asignación está precedido de alguno de estos operadores: *, /, %, +, -, <<, >>, &, |. Su estructura básica es la siguiente:

```
Expresion1 operador= Expresion2;
```

Donde:

- **Expresion1** es la primera expresión, que representa algún valor.

- **Operador** es alguno de los operadores antes mencionados.

- **Expresion 2** es la segunda expresión, que representa algún valor.

Un ejemplo puede ser:

```
Var += 100;
```

Donde lo que se está haciendo es sumar el valor 100 a Var. (Es igual que Var = Var + 100;)

- **Operador de incremento-decremento unitario**

Los operadores de incremento y decremento unitario son los que nos permiten aumentar o disminuir, dentro de una unidad, el valor de alguna variable.

Estos operadores pueden ser usados de tres formas diferentes: la primera es la simple; la segunda, la prefija; y la tercera, la postfija.

Forma simple: Se usa poniendo, antes del nombre de la variable, la expresión **++** (para hacer incremento unitario) o la expresión **--** (para hacer decremento unitario). Un ejemplo de incremento unitario sería el siguiente:

```
++Var;
```

Un ejemplo de decremento unitario sería así:

```
--Var;
```

Forma prefija: Se usa haciendo primero el incremento o decremento de la variable y luego, haciendo la operación que precede a este operando. Su estructura básica es la siguiente:

```
Variable1 operador  ++Variable2;
```

Un ejemplo al respecto lo constituiría la siguiente línea de código:

```
Var1 =  ++Var2;
```

En donde primero se incrementa en 1 la variable **Var2,** y posteriormente, se asigna su valor ya incrementado a la variable **Var1**.

Forma postfija: Se usa haciendo en primer lugar la operación que aparece antes que el operador, y luego gestionando el incremento o decremento de la variable. Su estructura básica es la siguiente:

```
Variable1 = Variable2++;
```

Un ejemplo al respecto sería la siguiente línea de código:

```
Var1 = Var2++;
```

En este caso, primero se le asigna el valor de la variable **Var2** a la variable **Var1** y luego se incrementa la variable **Var2** en 1.

- **Operadores relaciónales**

Se usan en las estructuras condicionales y repetitivas para comparar variables o expresiones:

Operador	Descripción
>	Se usa como Mayor Que...
<	Se usa como Menor Que
==	Se usa como Igual Que
>=	Se usa como Mayor o Igual Que
<=	Se usa como Menor o Igual Que
!=	Se usa como Diferente Que

- **Operadores lógicos**

Se usan para comparar dos expresiones y devuelven un valor verdadero, si la comparación es verdadera:

Operador	Descripción
||	Se usa como un OR
&&	Se usa como un AND
!	Se usa como un NOT
if (!x) Instrucción;	Si X es cero (falso), ejecuta la instrucción

- **Operadores aritméticos**

Se utilizan para realizar las operaciones aritméticas básicas. En las expresiones, primero se ejecutan las operaciones de multiplicación (*), división (/) y módulo (%), y después las sumas y las restas.

Operador	Descripción
+	Se usa para sumar
-	Se usa para restar
*	Se usa para multiplicar
/	Se usa para dividir, pero solo arroja el cociente
%	Se usa para dividir, pero solo arroja el residuo

- **Operador condicional**

Se usa para seleccionar una de las alternativas, separadas por dos puntos, dentro de una expresión incluida después de una comparación.

Operador	Descripción
?	Usa una de las alternativas separadas por dos puntos.

Un ejemplo podría darse si se compara la variable **x** con la variable **y**. Entonces, si **x** es menor que **y**, la variable **i** toma el valor 6; de lo contrario, toma el valor k +1:

```
i = (x < y ? 6 : k + 1);
```

- **¿Cómo asignar valores hexadecimales?**

Para hacer una asignación de un valor hexadecimal a una variable, se debe tener en cuenta que los valores hexadecimales siempre comienzan con los caracteres: 0x. La forma general es la siguiente:

```
Variable = 0xValorExadecimal;
```

Un ejemplo al respecto sería asignarle a la variable k el valor hexadecimal FF (que en decimales, equivale a 255).

```
k = 0xFF;
```

- **Variables como contadores**

Se da cuando a una variable se le suma un valor constante (normalmente el valor 1), y se usa principalmente para contar ciclos. Esto se hace asignándole a la variable contador, el valor de ella

misma, más el incremento. Este tipo de variables se deben inicializar en cero, antes de ser usadas. Su forma general es:

```
variable=variable+1;
```

También se puede usar con Operadores de Incremento Unitario, así:

```
variable++;
```

- **Variables como acumuladores**

Tiene lugar cuando se usa una variable para acumular los valores de otras variables. Esto se hace asignándole a la variable acumulador el valor de ella misma, más el valor de la variable que se desea acumular (aunque también podría ser el valor de una constante). Este tipo de variables se deben inicializar normalmente en cero, antes de usarlas, para que el resultado de la suma de los valores, al ser acumulados, resulte correcto.

```
NombreAcumulador = NombreAcumulador +variable;
```

Un ejemplo puede ser un acumulador llamado **sumatoria**, que haga la sumatoria de las edades de tres personas:

```
int sumatoria = 0;
int edad1 = 22;
```

```
        sumatoria = sumatoria+edad1;
int edad2 = 33;
        sumatoria = sumatoria+edad2;
int edad3 = 44;
        sumatoria = sumatoria+edad3;
println(sumatoria);
```

En este ejemplo, se inicializa la variable **sumatoria** con el valor **0**, luego se inicializa la variable **edad1** con el valor **22**, luego se suma el valor que tenga la variable **sumatoria** con el valor de la variable **edad1**; luego se inicializa la variable **edad2** con el valor **33**, se suma el valor que tenga la variable **sumatoria** con el valor de la variable **edad2** y se inicializa la variable **edad3** con el valor **44.** Posteriormente, se suma el valor que tenga la variable **sumatoria** con el valor de la variable **edad3** y, por último, se imprime el valor de la **sumatoria,** que debe dar **99**. En este ejemplo, la variable **sumatoria** fue acumulando los valores de las edades, en la medida en que las variables se iban inicializando.

- **Asignación de cadenas**

Para inicializar un arreglo de caracteres o para asignarle una secuencia de caracteres, no es necesario indicar el tamaño del vector, sino que se pone toda la secuencia entre corchetes y se ubica cada caracter entre comillas simples, agregando una coma para separar cada caracter. En el siguiente ejemplo, se pude

ver como se crea un arreglo de caracteres de cuatro posiciones:

```
char saludo[] = {'H', 'O', 'L', 'A'};
```

Para la asignación de una secuencia de caracteres a una cadena (tipo String), no es necesario indicar el tamaño del vector. Solo se debe poner el texto entre comillas:

```
String nombre = “Jose”;
```

Para asignarle a una cadena (tipo String) el contenido de un arreglo de caracteres, se usa la siguiente forma:

```
String NombreCadena = new String(NombreArregloCaracteres);
```

Un ejemplo puede ser asignarle el contenido del arreglo de caracteres, llamado **felino**, a la cadena tipo String, llamada **animal**:

```
char felino[] = {'G', 'A', 'T', 'O'};
String animal = new String(felino);
```

Cuando se muestre el contenido de la cadena **animal** en pantalla, aparecerá **GATO**.

1.5. Estructura básica de un programa

Cuando se programa, es fundamental saber que todas las instrucciones deben finalizar con punto y coma (;) y que hay una estructura básica que normalmente deben tener todos los programas, la cual se puede dividir en tres partes fundamentales:

- Comentarios
- Declaraciones externas
- Declaración de funciones

- **Comentarios**

Los comentarios son todas aquellas secciones de código que no son tenidas en cuenta en la ejecución del programa y que normalmente sirven para documentar el código y agregar explicaciones o descripciones. Los comentarios pueden contener cualquier clase de texto y pueden ponerse en cualquier parte del código. Existen dos formas de agregar comentarios: una para comentarios de una sola línea y otra, para comentarios en bloque.

Comentario de una sola línea: es aquel texto que esta seguido de la secuencia de caracteres // (que determinan el inicio del comentario) y el Enter al final de la línea de código, que determina el fin del comentario.

En el siguiente ejemplo, se hace uso de un comentario de una sola línea, creando la variable x de tipo int; y se inicializa en 0. Al final de la línea de código, aparece el comentario que describe lo que hace dicha línea:

```
int x=0; // La variable x se inicializa en 0
```

Comentario en bloque: Es todo aquello contenido dentro de la secuencia de caracteres: /* (que determina el inicio del comentario) y la secuencia final */ (que determina el fin del comentario).

En el siguiente ejemplo, se hace uso de un comentario en bloque: en la primera línea, se crea la variable x de tipo int, después, se encuentra el comentario en bloque que consta de tres líneas y, en la última línea de código, se inicializa x en 0.

```
int x;
/* Comentario en bloque
   Comentario en bloque
   Comentario en bloque */
x=0;
```

- **Declaraciones externas**

Las declaraciones externas permiten definir las librerías y las constantes que se usarán para la ejecución del programa, y se deben declarar primero que las funciones.

Definición de librerías

Las librerías son archivos que contienen segmentos de código que han sido programados de antemano, para facilitar la implementación de funciones de entrada y salida de datos, funciones de comunicaciones, funciones avanzadas de gráficos, etc.

Estas librerías normalmente son creadas por terceros. Algunas de ellas vienen incluidas en las distribuciones de Processing y otras deben descargarse manualmente.

Existen dos maneras para incluir estas librerías: en modo manual y en modo automático.

Modo manual: se escribe la palabra **import**, seguida del nombre de la librería, así:

```
import NombreLibrería.*;
```

Modo automático: se debe seleccionar la librería por su nombre, haciendo clic en el menú "**Sketch**" y luego en "**Import Library...**", y se selecciona la librería previamente instalada.

Si no está instalada la librería que se requiere, entonces se puede instalar haciendo clic en el menú "**Sketch**" y luego en "**Import Library...**". Luego, "**Add Library...**".

Hecho esto, se debe esperar un momento mientras se abre otra ventana, en la cual pueden buscarse librerías por nombre o por categorías. Entonces seleccionamos la librería que queramos instalar y hacemos clic en "**Install**".

Definición de constantes

Es un valor que no cambia dentro de todo la ejecución del programa, que puede ser numérico, alfanumérico o alfabético. Para definir una constante, debe seguirse esta estructura:

```
final TipoDato NombreConstante = Valor;
```

Un ejemplo sería declarar la constante PI con el valor decimal 3.14159

```
final float pi = 3.14159;
```

- **Declaración de funciones**

Las funciones, son segmentos de código en donde se definen una serie de instrucciones que se deben ejecutar cada vez que se llaman.

En Procesing, existen dos funciones principales: una de configuración inicial, que es: **setup();** y otra, la función principal, desde donde se llaman a todas las demás funciones, que es **draw()**.

El código de atras funciones, pueden ir en la misma pestaña del sketch en donde se encuentren las dos funciones principales (setup() y draw()) o pueden ir en una nueva pestaña (haciendo clic en el triángulo que aparece justo al lado de la pestaña, que tiene el nombre del sketch y seleccionando la opción crear).

Declaración función de configuración setup()

En la declaración de la función de configuración inicial setup() es en donde irá el código destinado a hacer las rutinas de inicialización o configuración del programa. Entre ellas, por ejemplo, establecer el tamaño de la ventana del programa, cargar las imágenes o cargar las fuentes. Su forma general es la siguiente:

```
void setup()
 {
     Instrucciones
      .
      .
      .
 }
```

Un ejemplo de podría ser declarar una función setup() que configure el tamaño de la ventana de la aplicación, a una resolución de 800 píxeles por 600 píxeles.

```
void setup() {
  size(800,600);
}
```

Declaración de la función principal draw()

En la declaración de la función principal draw() es en donde estará gran parte del código del programa y desde allí se llamarán las funciones definidas fuera de la función draw(). Su forma general es la siguiente:

```
void draw(argumentos)
{
        instrucciones
        .
        .
        .
}
```

Un ejemplo de podría ser declarar una función draw() que dibuje un cuadrado (de 10 píxeles por 10 píxeles) en la posición 20 en X y 50 en Y de la pantalla.

```
void draw() {
  rect(20,50,10,10);
}
```

1.7. Estructuras algorítmicas

Cuando se desarrollan programas, normalmente se recurre al uso de estructuras para llegar a la solución más concisa y ordenada de un problema. Las dos principales estructuras son las condicionales y las cíclicas.

- **Estructuras condicionales**

Las estructuras condicionales son aquellas que pueden ejecutar una serie de acciones, dependiendo si la condición pre-establecida es cierta o falsa. En otras palabras, es una estructura que comienza con una condición y para cuando la ejecución del programa llega a esa parte del algoritmo, dependiendo de lo que sea (cierta o falsa), se ejecuta una de las series de acciones o ninguna, (todo depende de cómo se establezca la condición). Existen varios tipos de estructuras condicionales. Las más usadas y comunes son: el IF simple, el IF compuesto y el switch (en caso de hacer).

Estructura Condicional: If Simple

Para usar esta estructura, se debe establecer una condición según la cual se ejecutarán determinadas acciones. Si para el momento en que la ejecución del programa llegue a la condición, esta última es verdadera, el programa ejecutará las instrucciones comprendidas entre los corchetes {} y saldrá de la estructura.

```
if (Condición)
{
  Instrucciones
}
```

Un ejemplo podría ser una aplicación para comprobar si la edad de una persona corresponde a un mayor de edad:

```
int edad=18;

void setup() {
  size(800,600);
}

void draw() {
  if(edad>17) {
    text("Usted es mayor de edad",10,100);
    }
 }
```

Estructura condicional: If Compuesta

Para usar esta estructura se debe establecer una condición según la cual se ejecutarán determinadas acciones. Si para el momento en que la ejecución del programa llegue a la condición, esta es verdadera, el programa ejecutará las instrucciones comprendidas entre los corchetes {} y saldrá de la estructura. Si es falsa, el programa ejecutará las instrucciones comprendidas entre los corchetes {}, que se encuentran después del **else** y saldrá de la estructura.

```
if (Condición)
 {
   Instrucciones
 } else {
   Instrucciones
 }
```

Un ejemplo podría ser una aplicación para comprobar si la edad de una persona corresponde a un mayor o a un menor de edad:

```
int edad=14;

void setup() {
  size(800,600);
  background(0); // se pone el fondo negro
}

void draw() {
  if(edad>18) {
    text("Usted es mayor de edad",10,100);
    } else {
    text("Usted es menor de edad",10,100);
    }
}
```

Estructura condicional switch (En caso de-hacer)

Para usar la estructura **switch** se debe establecer una variable, la cual será comparada con los valores que van seguidos de la palabra "**case**". Para cuando la ejecución del programa llegue a la estructura, si el valor "**case**" coincide con el de la variable de la estructura, entonces el programa ejecutará las instrucciones definidas entre las sentencias "**case**" y "**break**", y saldrá de la estructura. En caso de que el valor de la variable no coincida con ningún valor "**case**", entonces se ejecutarán las instrucciones establecidos entre las sentencias "**default**" y "**break**", y saldrá de la estructura.

```
switch(variable)
{
case Valor1:
    Instrucciones
    break;

case ValorN:
    Instrucciones
    break;

default:
    Instrucciones
    break;
}
```

Un ejemplo podría ser una aplicación que muestre en pantalla si se oprime una vocal con el teclado:

```
void draw() {
 if (keyPressed=true) {

    switch (key) {
    case 'a':
      background(125);
      text("a",10,10);
      break;
    case 'e':
      background (125);
      text("e",10,10);
      break;
    case 'i':
      background (125);
      text("i",10,10);
      break;
    case 'o':
      background (125);
      text("o",10,10);
      break;
    case 'u':
      background (125);
      text("u",10,10);
      break;
    default:
      background(125);
      text("Oprima vocales", 2,30);
      break;

    } // fin switch
  } // fin if
} // fin draw
```

- **Estructuras repetitivas**

Las estructuras repetitivas son aquellas que repiten la ejecución de una serie de acciones, hasta que no se cumpla una condición pre-establecida. La condición puede estar antes o después de la serie de acciones a repetir, dependiendo de la estructura repetitiva que se use.

Cuando se termina de ejecutar el conjunto de instrucciones establecidas en la estructura repetitiva, se regresa al inicio de la estructura para hacer un nuevo ciclo de repetición. Por lo tanto, se puede decir que una estructura repetitiva (aunque dentro de ella tuviera muchas más instrucciones), es considerada por el computador como una sola instrucción dentro del cuerpo del programa.

Estructura repetitiva do-while (Repetir-hasta que)

Para usar esta estructura se debe definir una condición que, cuando llega a ser verdadera, finaliza la repetición de ciclos. Es importante asegurarnos de que la condición pueda llegar a ser verdadera, ya que de no ser así, nuestro programa se volvería un loop infinito (es decir, que se repite infinitamente). Esto podría suceder debido a que la estructura **do-while** no tiene límite en el número de veces que se puede repetir el ciclo, puesto que este solo termina cuando la condición llega a ser verdadera. **Nota**: Es importante anotar que al

usar esta estructura, se ejecuta, como mínimo, una vez el ciclo. Si no se quiere esto, entonces se debe usar la estructura **while** y no la **do-while**.

```
do
{
  Instrucciones
}while(Condición);
```

Un ejemplo podría ser una estructura do-while para dibujar líneas horizontales cada 10 píxeles:

```
int i = 0;
 do {
  line(0, i, 100, i);
  i = i + 10;
 } while (i < 100);
```

Estructura repetitiva: while (Mientras-hacer)

Para usar esta estructura se debe definir correctamente la condición que, al ser, falsa finalizará la repetición de ciclos. Si la condición no llega a ser falsa, entonces nuestro programa se volvería un loop infinito (que se repite infinitamente). Esto puede suceder debido a que la estructura **while** no tiene límite en el número de veces en las cuales se puede repetir el ciclo, ya que solo termina cuando la condición llega a ser falsa. **Nota**: Es importante anotar que si se requiere ejecutar, como mínimo, una vez el ciclo, entonces se debe usar la estructura **do-while** y no la **while**.

```
while (Condición)
{
  Instrucciones
}
```

Un ejemplo podría ser una estructura while para dibujar líneas horizontales cada 10 píxeles:

```
int i = 0;
 while (i < 100) {
  line(0, i, 100, i);
  i = i + 10;
 }
```

Estructura repetitiva For (Para)

Esta estructura se define dentro de paréntesis (), donde se establecen tres partes que van separadas por punto y coma (;). En la primera de esas partes es en donde se inicializa la variable con la que se van a contar los ciclos. En la segunda, hay condición con la cual se determina si el valor de la variable supera el número de ciclos a realizar; mientras la tercera, se encarga de ir incrementando, ciclo a ciclo, el valor de la variable. Su forma general es la siguiente:

```
//      Inicialización    ;    Condición      ;      Incremento
 for ( Var = valor inicial ; Var <=> valor final ; Var = Var + incremento )
 {
   Instrucciones
 }
```

Para construir una estructura for se abre un paréntesis y se pone lo siguiente: primero se crea una variable con la que se van a contar los ciclos, que se inicializa -normalmente en 0 o en 1-. A continuación, se pone un punto y coma, y después se agrega la condición en donde se compara el valor de la variable, con el valor preestablecido, como el número de veces que se desea repetir el ciclo, seguido de otro punto y coma.

Por último, se incluye una expresión que se encarga de hacer el incremento de la variable, ciclo a ciclo, y se cierra el paréntesis. Normalmente, para el incremento se usa la expresión de incremento unitario. (ej. variable++). Después de esta, se ponen entre corchetes {} las instrucciones que se desea realizar en cada ciclo.

Un ejemplo podría ser una estructura for para dibujar líneas horizontales cada 10 píxeles:

```
for(int i=0; i<100; i=i+10){
  line(0, i, 100, i); // dibuja lineas cada 10 pixels
}
```

1.8. Creación y llamado de funciones

Una función es un conjunto de instrucciones, ordenadas para realizar una tarea concreta. Su uso permite reducir la repetición innecesaria de código, además de darle orden y facilitar su entendimiento y

optimización. Las funciones pueden requerir valores como argumentos y retornar valores al finalizar la ejecución de la función.

- **Definición de una función**

La estructura general para definir una función es la siguiente:

```
Tipo NombreFunción (Parámetros)
{
    Instrucciones
    .
    .
    .
}
```

Tipo: Indica la clase de valor que devuelve la función. En caso de que la función no devuelva ningún valor, se indica mediante la palabra void.

NombreFunción: Es un identificador de la función que será usado para hacer llamadas a la función. No debe ser repetido.

Parámetros: Componen la lista de argumentos de la función. Consta de un conjunto de variables con sus tipos asociados, separados por comas. Si la función no tiene argumentos, se indica con void.

NOTA: Dentro del cuerpo de una función no se puede definir otra.

Un ejemplo podría ser una función que se encargue de multiplicar dos datos, así:

```
int multiplica(int a, int b)
{
int r;
r = a*b;
return r;
}
```

En este caso, se usan dos argumentos, los cuales son los valores que se van a multiplicar y, al final de la función, se retorna el valor de la variable **r** (con la instrucción **return r;**), que almacena el resultado de la operación.

- **Llamada de una función**

Es la orden que hace que se ejecute la función. Su estructura general es la siguiente:

```
NombreFunción(parametro1,parametro2,... parametroN);
```

Cuando se efectúa la llamada a la función, el control se transfiere a esta para su ejecución, pasando al mismo tiempo los valores de sus argumentos.

La ejecución de la función finaliza cuando se llega al final del cuerpo o a una sentencia:

```
Return NombreVariable;
```

En este instante, el control es devuelto al punto de llamada y se retorna un valor que tenga almacenado, en ese instante, la variable (también se pueden usar valores constantes).

Continuando con el ejemplo de función (de la anterior sección), se puede llamar a la función **multiplica(),** dentro de la función draw(), así:

```
void draw()
{
multiplica(5,7);
}

int multiplica(int a, int b)
{
int r;
r = a*b;
println(r);
return r;
}
```

En este caso, al ejecutarse este código, se muestra en el área de consola el resultado de la multiplicación (35).

PARTE DOS:

ENTORNO DE PROGRAMACIÓN PROCESSING

- **¿Por qué es importante saber Processing?**

El lenguaje de programación Processing fue diseñado específicamente para hacer prototipado rápido de aplicaciones y su principal enfoque es ofrecerle a los artistas, diseñadores y creativos, un entorno de desarrollo amigable y versátil, que no exigiera demasiados conocimientos técnicos, para ser utilizado en la creación de aplicaciones interactivas.

Processing cuenta con una gran variedad de librerías (disponibles de forma libre y gratuita) para que los creadores puedan incorporar nuevas funcionalidades estéticas y técnicas a las aplicaciones interactivas que quieren desarrollar.

Este lenguaje está construido sobre JAVA, lo cual facilita la reutilización de código (Java), para integrarlo dentro de código Processing, sin tener que hacerle modificaciones sustanciales. Esto significa que el código escrito en Processing es multiplataforma, y que se puede ejecutar en en diferentes entornos como MAC OS, Windows, GNU/Linux o Android.

Además, el entorno de Processing ofrece la opción de exportar el código como JavaScript, para integrarlo dentro de páginas web. Esto se da como resultado de la filosofía que Sun Microsystems (Empresa creadora de JAVA) propuso en 1996: "Write once, run anywhere" y que en español puede traducirse como "Escribe una vez, ejecútalo en donde sea".

En resumen, las principales ventajas por las que se caracteriza Processing tienen que ver con su portabilidad, su comunidad activa, su variedad de librerías disponibles y su buena documentación[2], así como los foros[3], accesibles desde la página oficial de Processing.

- **Processing para programar Hardware**

Wiring es una plataforma Open Source de prototipado electrónico, creada por el colombiano Hernando Barragán en el año 2003. El entorno de desarrollo de Wiring está basado en el desarrollo de Processing, algo que permite una gran compatibilidad entre el hardware Wiring y las aplicaciones hechas con Processing.

Arduino es una plataforma de hardware abierto, que surge como una derivación del proyecto Wiring. Es 100% compatible con las aplicaciones que se hacen

---

2 http://processing.org/reference/
3 http://forum.processing.org/two/

con Processing. Solo se debe instalar la librería Firmata, tanto en el dispositivo Arduino como en la aplicación Processing, y luego se establece de forma directa la comunicación entre ambos.

- **Descarga e instalación de Processing**

Para usar Processing, el primer paso es descargarlo desde la página web oficial [4], en donde están disponibles las diferentes versiones para las plataformas Windows, Mac OS y GNU/Linux.

Existen dos versiones para cada plataforma, una con el JAVA incluido, que es de mayor tamaño para la descarga, y otra que no incluye JAVA, para usuarios expertos que quieran instalar el JAVA SDK [5] manualmente.

- **Prerequisitos**

Un aspecto importante, antes de comenzar a programar en Processing, es asegurarse de tener instalado el software adicional necesario para usarlo en otros modos de trabajo. En el caso particular del modo Android, se debe descargar e instalar el Android SDK Tools, desde la página oficial de desarrolladores de Android[6].

4http://processing.org/
5 http://oracle.com/technetwork/java/javase/downloads/index.html
6 http://developer.android.com/sdk/

La versión que se debe descargar es la llamada "SDK Tools Only", ya que esta contiene lo que necesita Processing para funcionar en modo Android.

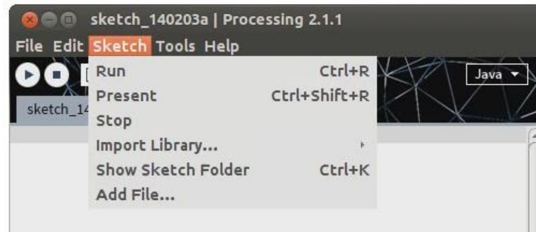
La versión "ADT Bundle" es mucho más grande e incluye el entorno de desarrollo Eclipse, que no es necesario para utilizar Processing.

- **Sketch**

Se denomina Sketch al código escrito dentro del entorno Processing. La palabra, traducida al castellano, significa "Bosquejo" o "Esquema", y tiene relación directa con la intención que tenían los creadores de Processing, de que este fuera un entorno de desarrollo rápido para prototipos de aplicaciones interactivas. Lo anterior significa que los Sketch son los bocetos de aplicaciones más grandes y complejas.

El Sketch se almacena en un archivo con extensión PDE, en una carpeta con el mismo nombre del Sketch. Es posible acceder a todos los Sketch desde el **Sketchbook**, en donde se encuentran todas las carpetas de cada Sketch creado. Para navegar el **Sketchbook** se debe hacer clic en la opción "**File**" del menú principal y después en "**Sketchbook**".

Para ingresar a la carpeta del Sketch se debe hacer clic en la opción "**Sketch**", del menú principal, y después en "**Show Sketch Folder**".



Otra manera de ingresar a la carpeta del Sketch, es presionando la combinación de teclas: **Crtl+K**, para abrirla de forma rápida.

La estructura de la carpeta del Sketch es la siguiente:

Archivos PDE: Son aquellos archivos que contienen el código fuente del Sketch.

Archivo AndroidManifest.xml (Archivo de configuración en Modo Android).

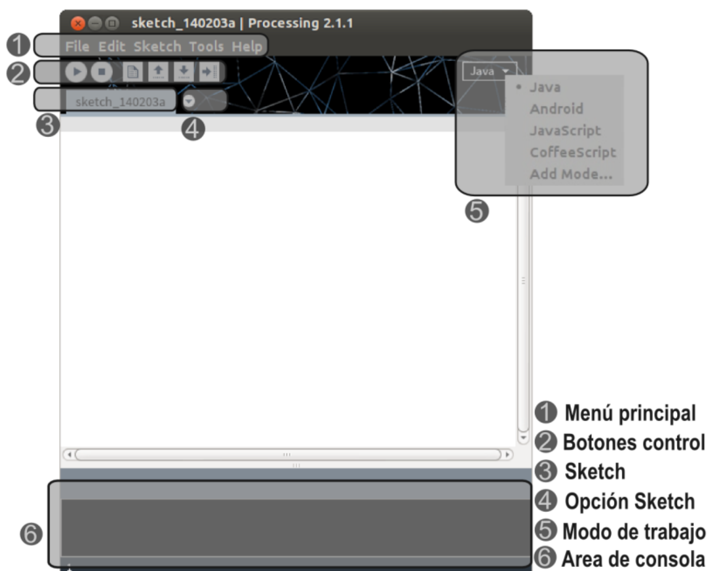
Archivo sketch.properties (Archivo de configuración).

Carpeta data: En esta carpeta se ponen los archivos que serán usados por el programa, como imágenes, fuentes, sonidos, archivos de texto, etc.

Carpeta code: En esta carpeta se ponen las librerías que serán usadas por el programa.

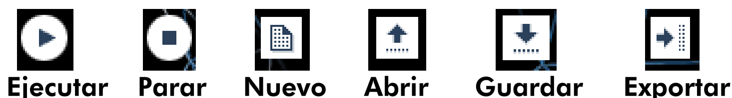
- **Interface**

Para comenzar a usar el entorno Processing, es fundamental conocer las diferentes partes que comprende la interfaz. En la siguiente imagen se puede observar las principales áreas de la interface:



En la parte superior de la ventana se encuentra el menú principal, que puede variar según el modo de trabajo que se tenga seleccionado (Por defecto, el modo es Java). Se puede cambiar el modo, haciendo clic en el área marcada con el número cinco en la anterior imagen. Entre los diferentes modos que se pueden elegir está Java, Android, JavaScript y CoffeeScript.

Debajo del menú principal aparecen una serie de botones que sirven de controles para el Sketch, los cuales tienen las siguientes funciones:



Se pueden observar, debajo de los botones de control, las pestañas de los Sketch (como mínimo se debe ver una pestaña), y después de la última pestaña hay un triángulo con el cual se accede a las opciones para crear y eliminar pestañas, y para seleccionar la pestaña que se desea modificar.

- **¿Cómo acceder a la información de referencia de las primitivas?**

Para acceder a la información de referencia de cada una de las primitivas (palabras reservadas) en Processing, se debe seleccionar la palabra reservada (usualmente está en otro color) y, con el botón derecho del mouse, hacer clic en donde dice: "**Find in Reference**". Esto, automáticamente, abrirá el navegador de internet en donde se mostrará toda la información relacionada con dicha palabra, reservada de la API de Processing. Nota, una referencia que es un archivo de Processing, que no necesita de Internet. Si por el contrario no se recuerda una palabra reservada, se pueden consultar, en la Parte cuatro de este libro, sección "Lista de palabras reservadas del lenguaje

Processing”. Otra opción es escribir: **rect();** en un sketch vacío y seleccionar la palabra rect. Luego, hacer clic en “**Find in Reference**” y en la página que abre el navegador, dar clic donde dice: “**Language**”, opción que está ubicada en la parte izquierda, como la primera del menú de la página. Allí aparecen todas las palabras reservadas, clasificadas por su tipo.

- **¿Cómo acceder a los ejemplos?**

Para acceder a los ejemplos que trae Processing se debe hacer clic en el menú “**File**” y luego en “**Examples…**”. Se espera un momento mientras, se abre otra ventana, en la cual aparecen un muy buen número de ejemplos, clasificados por temas.

- **¿Cómo instalar una librería?**

Para incluir una de estas librerías se deben seleccionar por el nombre, haciendo clic en el menú “**Sketch**”, luego en “**Import Library…**” y finalmente en “**Add Library…**”. Hecho esto, se espera mientras se abre otra ventana, en la cual podremos buscar librerías por nombre o categoría. Entonces se selecciona la librería que se quiera instalar y se da clic en “**Install**”.

PARTE TRES:

PROTOTIPADO DE APLICACIONES INTERACTIVAS EN PROCESSING

3.1. Consejos antes de comenzar

- **Indentar el código:**

Es una práctica que se debe tomar como hábito cuando se va a programar, ya que hace más fácil la lectura y la comprensión del código, tanto para el mismo autor como para otros programadores. Consiste en insertar espacios antes de las líneas de código, que hagan parte de una función, o de una estructura algorítmica. Un ejemplo de código sin indentar sería:

```
void setup () {
noLoop();
}
void draw () {
int x, y;
for (x = 0; x <= 10; x++) {
for (y = 0; y <= 10; y++) {
println(x,y);
}
}
}
```

Indentado este ejemplo, se ve claramente qué queda ubicado dentro de cada parte, por lo que resulta más fácil de entender.

```
void setup () {
  noLoop();
}

void draw () {
  int x, y;
  for (x = 0; x <= 10; x++) {
      for (y = 0; y <= 10; y++) {
         println(x,y);
      }
  }
}
```

- **Documentar y comentar el código:**

Es otra práctica fundamental para incorporar en los hábitos de un buen programador, que consiste en crear un archivo de documentación en el que se relacione la versión del Sistema Operativo, la del compilador de Processing y la de las librerías externas usadas para crear cada aplicación, además de documentar cómo se usa cada librería. Documentar el código es una inversión a largo plazo, ya que es muy usual que se necesite reciclar código para otros programas, pero si no está bien documentado, puede resultar más difícil interpretarlo que crearlo desde cero.

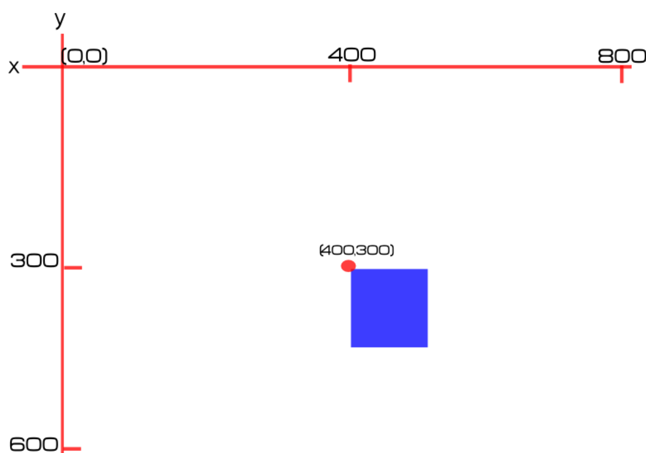
3.2. Métodos básicos de salida de en Processing

En esta sección, se exploran las diferentes formas de generar salidas de información, ya sea a través de la pantalla o de los parlantes.

- **¿Cómo funciona la pantalla en Processing?**

El sistema de coordenadas de pantalla en Processing comienza en el punto (0,0) el cual está ubicado en la esquina superior izquierda de la pantalla. Los valores de las coordenadas, en X, se incrementan hacia la derecha; y los valores de las coordenadas en Y, se incrementan hacia abajo.

Así pues, en una pantalla con una resolución de 800x600 píxeles, la posición (400,300) sería la que se muestra en la siguiente gráfica:



- **¿Cómo activar el modo de pantalla completa?**

Para hacer que, cuando se ejecute el programa, la ventana se abra en modo pantalla completa, se debe usar el siguiente código dentro de la función setup():

```
void setup()
{
size(displayWidth, displayHeight);
}
```

- **¿Cómo mostrar texto en pantalla?**

Para mostrar texto en pantalla, se puede recurrir a varias opciones. La forma más simple es la siguiente:

```
text("Hola", 15, 45);
```

En este caso, se imprime el texto Hola en la posición (15,45) de la ventana. Sin embargo, se usa una fuente genérica por defecto.

Para seleccionar una fuente particular para el texto, se puede hacer de forma rápida en dos líneas de código, si bien consume muchos más recursos del sistema:

```
textFont(createFont("Arial",40));
text("Hola", 15, 45);
```

Una forma más larga, pero que consume menos recursos, es la siguiente:

```
PFont mifuente;
mifuente = loadFont("Arial.vlw");
textFont(mifuente, 22);
text("Hola", 10, 50);
```

Primero se debe crear la fuente que se va a usar; en este caso se debe hacer clic en el menú principal, en donde dice: "**Tools**"; y luego en "**Create Font...**". Después se abrirá otra ventana en la cual podemos seleccionar la fuente, el tamaño y el nombre del archivo, el cual tendrá como extensión **.vlw** y se almacenará en la carpeta **data** del Sketch. Hecho esto, en el código de nuestro programa, se creará la variable mifuente de tipo PFont, en la cual se cargará la fuente deseada. Después, se carga con la directiva **loadFont()** el archivo de la fuente que se generó inicialmente (en este caso Arial.vlw). A continuación, se carga la configuración de la fuente con la instrucción **textFont**, que tiene dos parámetros. Primero, la variable de tipo PFont y después, el tamaño de fuente en el que se desea presentar el texto. Por último, se muestra el texto "Hola" en la posición (10,50) de la ventana.

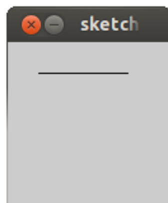
- **¿Cómo dibujar con primitivas 2D?**

Processing cuenta con las primitivas: line, rect, quad, triangle y ellipse para dibujar formas 2D en pantalla.

Líneas

La primitiva line() se usa para dibujar líneas. Los dos primeros valores definen el punto donde inicia la línea y los últimos dos valores, definen el punto donde esta finaliza:

```
line(10,10,80,10);
```

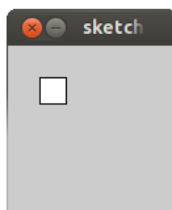


En el ejemplo anterior, se dibuja una línea desde el punto (10,10) hasta el punto (80,10) de la ventana.

Cuadrados

La primitiva rect()se usa para dibujar cuadrados. Los dos primeros valores definen la posición de la esquina superior izquierda del cuadrado, el tercer valor define el tamaño del cuadrado en el eje X y el último valor define el tamaño del cuadrado en el eje Y:

```
rect(10,10,20,20);
```

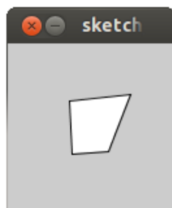


En el ejemplo anterior, se dibuja un cuadrado de un tamaño de 20 píxeles por 20 píxeles en la posición (10,10) de la ventana.

Cuadriláteros

La primitiva quad() se usa para dibujar cuadriláteros. Los dos primeros valores definen la posición del primer punto del cuadrilátero, los dos siguientes valores definen la posición del segundo punto del cuadrilátero, los dos siguientes valores definen la posición del tercer punto del cuadrilátero y los dos últimos valores definen la posición del último punto del cuadrilátero, así:

```
quad(33, 30, 80, 25, 63, 69, 35, 71);
```

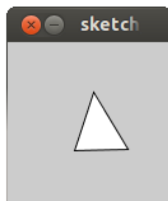


En el ejemplo anterior, se dibuja un cuadrilátero entre los puntos (33,30), (80,25), (63,69) y (35,71) de la ventana.

Triángulos

La primitiva triangle() se usa para dibujar triángulos. Los dos primeros valores definen la posición del primer punto del triángulo, los dos siguientes determinan la posición del segundo punto del triángulo y los dos últimos definen la posición del último punto del triángulo:

```
triangle(38, 71, 53, 25, 80, 70);
```

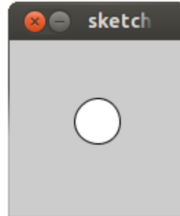


En el ejemplo anterior, se dibuja un triángulo entre los puntos (38,71), (53,25), y (80,70) de la ventana.

Elipses

La primitiva ellipse() se usa para dibujar elipses. Los dos primeros valores definen la posición del centro de la elipse, el tercer valor define el tamaño de la elipse en el eje X y el último valor define el tamaño de la elipse en el eje Y, así:

```
ellipse(50, 45, 33, 33);
```



En el ejemplo anterior, se dibuja una elipse de un tamaño de 33 píxeles por 33 píxeles en la posición (50,10) de la ventana.

- **Manejo del color**

Processing cuenta con las primitivas: background, fill y noFill para manejar el color del fondo tanto de las formas y fuentes como de la ventana.

Color fondo

Se usa la función background(), que se puede llamar con un argumento para manejar tonos, en una escala de grises, con valores entre 0 y 255:

```
background(125);
```

O se puede hacer con tres argumentos para manejar colores, en formato RGB, así:

```
background(0,0,255);
```

Color de relleno

Se usa la función fill() para definir el color de relleno de las formas o los textos, y al igual que background(), puede usarse con un argumento para manejar tonos, en una escala de grises, con valores entre 0 y 255:

```
fill(125);
```

Otra forma de hacerlo es con tres argumentos, para manejar colores en formato RGB, así:

```
fill(0,0,255);
```

Sin relleno

Se usa la función noFill() para definir que no haya color de relleno en las formas o en los textos.

```
noFill();
```

- **Manejo de bordes**

Color borde

Se usa la función stroke() para definir el color de los borde Con tres argumentos, puede hacerse para manejar colores en formato RGB:

```
stroke(0,0,255);
```

Sin borde

Se usa la función noStroke() para definir que no haya borde de relleno en las formas o en los textos, así:

```
noStroke();
```

Ancho borde

Se usa la función strokeWeight() para definir el grosor del borde en las formas o en los textos, así:

```
strokeWeight(valor);
```

- **¿Cómo mostrar imágenes en la pantalla?**

En Processing se pueden cargar imágenes en varios formatos (.gif, .jpg, .tga, y .png). Un ejemplo de cómo cargar y mostrar imágenes en pantalla, podría ser el siguiente:

```
// Se declara la variable varima para almacenar imágenes
PImage varima;
// Se carga la imagen en la variable varima
varima = loadImage("foto.png");
// Se muestra en pantalla la imagen
image(varima,100,50);
```

En este ejemplo se crea primero la variable "varima", de tipo **PImage**, luego se carga la imagen "foto.png",

que debe estar guardada en la carpeta **data** del sketch. Y por último, muestra la imagen en la posición (100,50) de pantalla, mediante la instrucción image().

- **¿Cómo reproducir videos?**

Processing puede reproducir videos en formato **MOV.** Sin embargo, la plataforma GNU/Linux permite reproducir cualquier tipo de video. El video que se va a reproducir se debe guardar en la carpeta **data** del Sketch.

```
import processing.video.*;
Movie video;

void setup() {
  size(800, 600);
  background(0);
  video = new Movie(this, "video.mov"); // carga el video
  video.loop(); // reproduce el video en loop
}
void draw() {
  image(video, 0, 0, width, height);
}
void movieEvent(Movie buffer) {
  buffer.read();
}
```

- **Cómo reproducir un sonido**

Para reproducir un sonido se usa la librería: "**Minim**", la cual está incluida por defecto dentro del entorno de programación Prosessing versión 2.1.1. Un ejemplo podría ser el siguiente:

```
import ddf.minim.*;
 Minim audio;
 AudioPlayer cancion;

void setup()
 {
    size(800, 600);
    audio = new Minim(this);  // inicializa el objeto tipo Minim
    cancion = audio.loadFile("cancion.mp3");  // carga el archivo
 }

void draw()
 {
    if (keyPressed == true) cancion.play();
 }

void stop()
 {
   cancion.close();  // detiene la reproducción del audio
   audio.stop();  // detiene el objeto Minim
   super.stop();
 }
```

Nota: La canción debe estar guardada en la carpeta **data** del sketch.

3.3. Métodos básicos de entrada

En esta sección, se exploran diferentes ejemplos que permiten acceder a diferentes periféricos de entrada, como son los teclados, los ratones, los micrófonos y las cámaras web.

Nota: Para utilizar los diferentes métodos de entrada y salida de texto, se deben conocer primero las secuencias escape, que se explican en la parte cuatro de este manual.

- **RATÓN**

A continuación se presentan las diferentes funciones disponibles, en Processing, para acceder a los estados del ratón, durante la ejecución del programa.

Estado de mousePressed

Permite reconocer si se ha presionado algún botón del ratón, sin tener que declarar una función adicional. Código de ejemplo:

```
// Si se hace clic se mueve el cuadrado
int x=0;
void draw() {
   background(210);
   rect(x,45,10,10);
```

```
  if (mousePressed == true) x++;
  if (x>89) x=0;
}
```

En este ejemplo, si se presiona cualquier botón del ratón, entonces se cambia el tono de relleno del cuadrado, con blanco o negro según sea el caso.

Función mousePressed()

Esta es una función que se puede declarar en Processing, para reconocer si se ha presionado algún botón del ratón. Código de ejemplo, como función:

```
float fondo = 255;

void setup(){
  size(800, 600);
  fill(125);
}

void draw(){
  background(fondo);
  text("Oprima un botón del ratón para cambiar el fondo", 20, 30);
}

void mousePressed() {
  fondo=fondo*-1;
}
```

En el ejemplo anterior, se cambia el fondo de la ventana cada vez que se presiona algún botón del ratón.

Función mouseReleased()

Esta función permite reconocer si se ha dejado de presionar algún botón del ratón, sin importar si se ha movido desde que se hace clic. Código de ejemplo:

```
float fondo = 255;

void setup(){
  size(800, 600);
  fill(125);
}

void draw(){
  background(fondo);
  text("Oprima un botón del ratón después cambiará el fondo", 20, 30);
}

void mouseReleased () {
  fondo=fondo*-1;
}
```

En este ejemplo se cambia el fondo de ventana justo después se deje de presionar el botón del ratón. Si se deja presionado el botón no pasa nada, pero cuando se suelta el botón entonces sucede el cambio.

Estado de mouseButton

Esta función permite reconocer cual botón del ratón se ha presionado. Código de ejemplo:

```
void draw() {
   background(210);
   if (mousePressed  == true){
     if (mouseButton == LEFT) rect(10,10,10,10);
     if (mouseButton == RIGHT) ellipse(20,20,10,10);
   }
 }
```

En este ejemplo se reconoce el botón presionado. Si se oprime el botón derecho, se dibuja un círculo, y si se presiona el botón izquierdo, entonces se dibuja un cuadrado.

Función mouseClicked()

Esta función es llamada justo después de que un botón del ratón ha sido presionado y luego soltado, pero solo funciona si la posición del mouse no cambia desde que se hace clic. Código de ejemplo:

```
float fondo = 255;

void setup(){
  size(800, 600);
  fill(125);
}
```

```
void draw(){
  background(fondo);
  text("Oprima un botón del ratón después cambiará el fondo", 20, 30);
}

void mouseClicked() {
  fondo=fondo*-1;
}
```

Estado de mouseX

Esta función permite conocer la posición actual del ratón, en el eje X. Código de ejemplo:

```
void draw()
 {
   background(210);
   ellipse(50, 50, mouseX, 80);
 }
```

En este ejemplo se incrementa o disminuye el ancho de la elipse, de acuerdo la posición del ratón en el eje X.

Estado de mouseY

Esta función permite conocer la posición actual del ratón en el eje Y. Código de ejemplo:

```
void draw()
 {
   background(210);
```

```
   ellipse(50, 50, 80, mouseY);
}
```

En este ejemplo se incrementa o se disminuye el alto de la elipse, de acuerdo a la posición del ratón en el eje Y.

Función mouseDragged()

Esta función permite reconocer si se está moviendo el ratón con algún botón presionado. Código de ejemplo:

```
int x=0,y=0;
void draw(){
   background(210);
   rect(x,y,10,10);
 }
void mouseDragged() {
   x=mouseX();
   y=mouseY();
}
```

En este ejemplo, si el ratón se mueve con algún botón presionado, entonces el cuadrado se desplaza con el ratón. Cuando se dejan de oprimir los botones del ratón, entonces el cuadrado no se mueve.

Función mouseMoved()

Esta función permite reconocer si se está moviendo el ratón sin estar presionado ningún botón. Código de ejemplo:

```
int i = 0;
  void setup() {
  frameRate(5);
 }
 void draw() {
  background(210);
  rect(50, i*15, 10, 10);
 }
 void mouseMoved() {
  i++;
  if (i > 5) {
    i = 0;
   }
 }
```

En este ejemplo, si se mueve el ratón en cualquier dirección, el cuadrado se desplaza 15 pixeles hacia abajo. En cambio, si este llega al límite de la ventana, el cuadrado vuelve a la parte superior de la misma.

Estado de pmouseX

Esta función permite conocer la posición del ratón en el eje X del frame anterior. En otras palabras, sirve para saber cuál era su posición en el eje X, antes de llegar a la posición actual.

Esto permite determinar la dirección en que se está moviendo el ratón. Si la posición anterior en el eje X es menor a la posición actual, significa que el ratón se

está moviendo de izquierda a derecha. Si la posición anterior en el eje X es mayor a la posición actual, quiere decir que el ratón se está moviendo de derecha a izquierda.

Un ejemplo podría ser que un cuadrado se mueva en la misma dirección en la que se mueva el ratón, sobre el eje X dentro la ventana.

```
int direccion=1, x=0;

void draw() {
  background(204);
  if (mouseX > pmouseX) direccion= 1;
  if (mouseX < pmouseX) direccion = -1;
  switch(direccion) {
  case 1:
    x++;
    if (x>89) x=0;
    break;
  case -1:
    x--;
    if (x<11) x=90;
    break;
  }
  rect(x,45,10,10);
 }
```

Estado de pmouseY

Esta función permite conocer la posición del ratón en el eje Y del frame anterior. En otras palabras, sirve para saber cuál era su posición en el eje Y, antes de llegar a la posición actual.

Esto permite determinar la dirección en la que se está moviendo el ratón. Si la posición anterior en el eje Y, es menor a la posición actual, significa que el ratón se está moviendo de arriba hacia abajo. En cambio, si la posición anterior en el eje X, es mayor a la posición actual, significa que el ratón se está moviendo de abajo hacia arriba.

Un ejemplo podría ser que un cuadrado se mueva en la misma dirección en la que se mueva el ratón, sobre el eje Y dentro la ventana.

```
int direccion=1, y=0;

void draw() {
  background(204);

  if (mouseY > pmouseY) direccion= 1;
  if (mouseY < pmouseY) direccion = -1;

  switch(direccion) {
  case 1:
    y++;
```

```
      if (y>89) y=0;
      break;
   case -1:
      y--;
      if (y<11) y=90;
      break;
   }
   rect(45,y,10,10);
 }
```

- **TECLADO**

A continuación, se presentan las diferentes funciones disponibles en Processing, para acceder a los estados del teclado durante la ejecución del programa.

Función keyPressed

Esta función permite reconocer si se ha presionado alguna tecla. Código de ejemplo:

```
// Si se oprime cualquier tecla se mueve el cuadrado
int x=0;
void draw() {
  background(210);
  rect(x,45,10,10);
  if (keyPressed == true) x++;
  if (x>89) x=0;
 }
```

En este caso, si se presiona cualquier tecla, entonces el cuadrado se mueve sobre el eje X.

Función key

Esta función permite conocer cuál fue de la última tecla presionada. Normalmente, se usa para hacer comparaciones y determinar si se ha presionado una tecla específica. Código de ejemplo:

```
void setup(){
  size(800, 600);
  fill(125);
}

void draw() {
  text("Oprima n para fondo negro o b para fondo blanco ",20,30);
  if(keyPressed) {
  if (key == 'n' || key == 'N') background(0);
  if (key == 'b' || key == 'B') background(255);
  }
}
```

En este ejemplo, cambia el fondo de la ventana si se oprime la tecla n o la tecla b, según sea el caso.

Función KeyCode

Esta función permite conocer el código de la última tecla presionada. Se usa, especialmente, para reconocer las teclas codificadas. Código de ejemplo:

```
int x=40;

void draw() {
   background(0);
   rect(x,40,10,10);
}

void keyPressed(){
if (key == CODED)
{
   switch(keyCode) {
   case LEFT:
      x--;
      if (x>89) x=0;
      break;
   case RIGHT:
      x++;
      if (x<11) x=90;
      break;
   } // fin switch
 } // fin if
}
```

Se hace uso de las teclas codificadas **RIGHT** y **LEFT en este ejemplo.** Con ellas se controla el movimiento de un rectángulo, en el eje X, dentro de la ventana. Para hacer uso de las teclas codificadas, se le debe agregar al código la estructura condicional **if (key == CODED),** que permite filtrar las teclas que son

codificadas para, posteriormente, hacer las respectivas comparaciones, usando los códigos que se presentan a continuación:

Códigos de las teclas

Teclas Codificadas	Teclas No Codificadas
UP	BACKSPACE
DOWN	TAB
LEFT	ENTER
RIGHT	RETURN
ALT	ESC
CONTROL	DELETE
SHIFT	

Función keyPressed()

Esta función permite reconocer si se ha presionado alguna tecla. Código de ejemplo:

```
float fondo = 255;

void setup(){
   size(800, 600);
   fill(125);
}

void draw(){
   background(fondo);
   text("Oprima un botón del ratón para cambiar el fondo", 20, 30);
}
```

```
void keyPressed() {
  fondo=fondo*-1;
}
```

En este ejemplo cambia el fondo de la ventana cada vez que se presiona alguna tecla.

Función keyReleased()

Esta función permite reconocer si se ha dejado de presionar alguna tecla. Código de ejemplo:

```
float fondo = 255;

void setup(){
  size(800, 600);
  fill(125);
}

void draw(){
  background(fondo);
  text("Oprima un botón del ratón después cambiará el fondo", 20, 30);
}

void keyReleased () {
  fondo=fondo*-1;
}
```

En el ejemplo, se cambia el fondo de ventana justo después se deje de presionar la tecla. Cuando se presiona la tecla no pasa nada, pero cuando se suelta entonces sucede el cambio.

- **MICRÓFONO**

Para acceder al micrófono se hace uso de la librería: "**Minim**", la cual está incluida por defecto en el entorno de programación Processing, versión 2.1.1. Un ejemplo de cómo usar el micrófono puede ser el siguiente:

```
import ddf.minim.*;
Minim entrada;
AudioInput microfono;
// valor para subir o bajar sensibilidad del micrófono
float sensibilidad = 0.01;

void setup() {
  size(500, 300, P3D);
  // inicializa el objeto tipo Minim
  entrada = new Minim(this);
  // inicializa el microfono
  microfono = entrada.getLineIn();
  // activa altavoz microfono (borre esta línea para silenciarlo)
  microfono.enableMonitoring();
}

void draw(){
  background(0);
  if (microfono.left.get(1)> sensibilidad)
  {
    ellipse(width/2, height/2, 55, 55);
  } else {
```

```
        text("Hable más duro para mostrar una elipse",10,20);
    }
}
```

- **WEBCAM**

Para acceder a la webcam, se hace uso de la librería: "**Video**", la cual está incluida por defecto dentro del entorno de programación Processing, versión 2.1.1. Un ejemplo[7] de cómo usar una webcam puede ser el siguiente:

```
import processing.video.*;
Capture camara;

void setup() {
  size(800, 600);
  camara = new Capture(this);
  camara.start();
}

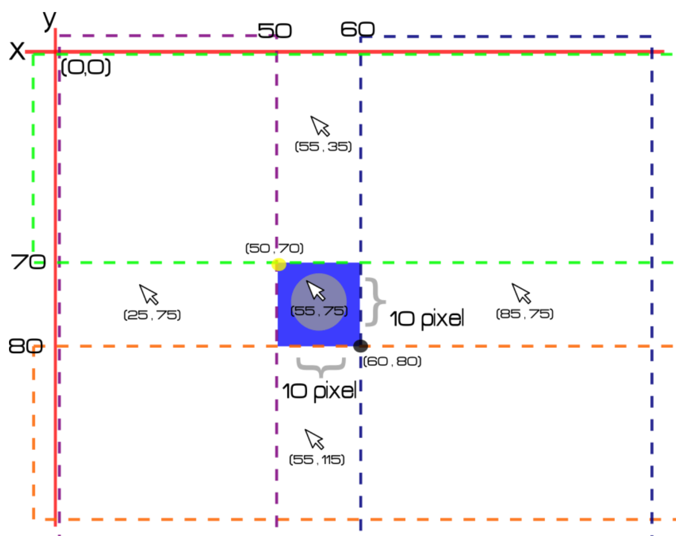
void draw() {
  if (camara.available() == true)
      {
         camara.read();
      }
  image(camara, 0, 0);
}
```

[7] Basado en: http://processing.org/reference/libraries/video/Capture_start_.html

3.4. Interacción y movimiento

- **¿Cómo crear botones?**

Para crear un botón se deben considerar cuatro condiciones básicas que permiten determinar si la posición del cursor del ratón se encuentra dentro del área del botón. La primera es determinar si la posición del ratón en el eje X, es mayor a la posición X en donde inicia el botón (punto amarillo). La segunda, definir si la posición del ratón en el eje X es menor a la posición X en donde termina el botón (punto gris). En tercer lugar, se determina si la posición del ratón en el eje Y es mayor a la posición Y en donde inicia el botón (punto amarillo). Finalmente, la cuarta condición pasa por determinar si la posición del ratón en el eje Y es menor a la posición Y en donde termina el botón (punto gris).



En la Figura 1 se puede observar que la sección punteada de color **violeta** es la que se puede descartar, si la posición del ratón cumple con la **primera** condición. Es decir que el ratón está en cualquier lugar, menos en esa área. La sección punteada de color **azul** es la que se puede descartar, si la posición del ratón cumple con la **segunda** condición. La sección punteada, de color **verde,** es la que se puede descartar, si la posición del ratón cumple con la **tercera** condición. La sección punteada de color **naranja** es la que se puede descartar, si la posición del ratón cumple con la **cuarta** condición. Un ejemplo de esto podría ser:

```
void setup() {
    size(displayWidth, displayHeight);
    background(0);
 }

  void draw(){
    text("Hacer clik en cada cuadrado ", 10, 20);
    rect(50,70,10,10);
    rect(80,70,10,10);
  }

void mousePressed(){
  if((mouseX > 50) &&(mouseX < 60 )&&(mouseY > 70) &&(mouseY < 80))
  {
     ellipse(55,50,10,10);
     return;
  }

  if((mouseX > 80) &&(mouseX < 90 )&&(mouseY > 70) &&(mouseY < 80))
  {
```

```
        ellipse(85,50,10,10);
        return;
    }
} // fin mousePressed()
```

En este ejemplo, los botones son de 10 pixeles de ancho por 10 pixeles de alto. En la primera condición de la estructura if (mouseX>50), se compara la posición en X de ratón con la posición en X, donde inicia el primer botón (en este caso 50).

En la segunda condición (mouseX<60), se compara la posición en X de ratón con la posición en X donde termina el primer botón (en este caso 60, porque el botón es de 10 pixels).

En la tercera condición de la estructuras if (mouseY>70), se compara la posición en Y, del ratón, con la posición en Y donde inicia el primer botón (en este caso 70).

En la cuarta condición (mouseY<80), se compara se compara la posición en Y del ratón, con la posición en Y donde termina el primer botón (en este caso 80, porque el botón es de 10 pixeles).

Si la posición del ratón cumple con estas cuatro condiciones, significa que este se encuentra en el área del botón, y entonces se dibuja un círculo encima de cada cuadrado sobre el que se haga clic.

Lo anterior se puede evidenciar claramente en la siguiente tabla, donde se analizan diferentes valores:

POS. RATÓN EN X	POS. RATÓN EN Y	MOUSEX>50	MOUSEX<60	MOUSEY>70	MOUSEY<80	RESULTADO
25	75	NO	SI	SI	SI	NINGUNO
55	35	SI	SI	NO	SI	NINGUNO
55	**75**	**SI**	**SI**	**SI**	**SI**	**DIBUJA CIRCULO**
85	75	SI	NO	SI	SI	NINGUNO
55	115	SI	SI	SI	NO	NINGUNO

Nota: Los botones pueden ser figuras dibujadas en pantalla (como en el código anterior), pero también ser imágenes, para lo cual es necesario conocer su tamaño en pixeles, para calcular correctamente las condiciones del botón. Además, se debe recordar que la imagen debe estar guardada en la carpeta **data** del sketch.

- **¿Cómo mover imágenes en pantalla, con el teclado?**

En el siguiente ejemplo se usan dos variables para almacenar las posiciones en los ejes x, y de la imagen. Y se usa la función **keyPressed()** para que, de acuerdo a la flecha del teclado que se oprima, entonces se incremente o decrezca en 10 pixeles alguna de las dos variables.

```
PImage ima; // Se crea variable para cargar la imagen
int x=350; // variable de la posición en X de la imagen
int y=250; // variable de la posición en Y de la imagen
```

```
void setup() {
background(255);
size(800,600);
ima = loadImage("logo.gif"); // se carga la image
}

void draw() {
background(255); // pone el fondo de color blanco
image(ima, x, y); // muestra la imagen en la posiscion x,y
}

void keyPressed(){
if (key == CODED)
{
  switch(keyCode) {
   case UP:
      y=y-10;
      break;
   case DOWN:
      y=y+10;
      break;
   case LEFT:
      x=x-10;
      break;
   case RIGHT:
      x=x+10;
      break;
  } // fin switch
 } // fin if
}
```

Nota: No olvidar que la imagen debe estar guardada en la carpeta **data** del sketch.

- **¿Cómo hacer pequeñas animaciones?**

En el siguiente ejemplo se usan dos variables para almacenar las posiciones en los ejes x, y de la imagen. Y en la función **draw()**, se hace uso de dos condiciones para aplicar los incrementos en dichas variables. De esta forma, se generar el movimiento de la imagen. Además, dentro de la función setup(), se hace uso de la instrucción **frameRate(15);** con la cual se controla la velocidad de la animación.

```
PImage ima; // Se crea variable para cargar la imagen
int x=0; // Variable de la posición en X de la imagen
int y=0; // Variable de la posición en Y de la imagen

void setup() {
  size(800,600);
  ima = loadImage("logo.gif"); // Carga la imagen
  frameRate(15); // Selecciona la velocidad de la animación
  fill(0);
}

void draw() {
  background(255); // Pone el fondo de color blanco
  image(ima, x, y); // Muestra la imagen en la posición x,y
  if(x<750) x=x+10; // incrementa en 10 la posición en X
```

```
    else
      if(y<550) y=y+10; // incrementa en 10 la posición en Y
        else text("Fin Animación ",350,300);
}
```

Nota: Recordad que la imagen debe estar guardada en la carpeta **data** del sketch.

- **¿Cómo controlar, con el teclado, la dirección de una animación?**

En el siguiente ejemplo se usan dos variables para almacenar las posiciones en los ejes x, y de la imagen. Se usa la función **keyPressed()** para que, de acuerdo a la flecha del teclado que se oprima, se defina la dirección en la cual debe moverse la imagen.

```
PImage ima; // Se crea variable para cargar la imagen
int x=400; // Variable de la posición en X de la imagen
int y=300; // Variable de la posición en Y de la imagen
int direccion=0;

void setup() {
  size(800,600);
  ima = loadImage("logo.gif"); // Carga la imagen
  frameRate(15); // Selecciona la velocidad animación
}

void draw() {
  background(255); // Pone el fondo de color blanco
```

```
      switch (direccion){
        case 1:
          y=y-10;
          break;
        case 2:
          y=y+10;
          break;
        case 3:
          x=x-10;
          break;
        case 4:
          x=x+10;
          break;
      } // fin switch
  image(ima, x, y); // Muestra la imagen en la posición x,y
}

void keyPressed(){
  if (key == CODED)
  {
      switch (keyCode){
        case UP:
          direccion = 1;
          break;
        case DOWN:
          direccion = 2;
          break;
        case LEFT:
          direccion = 3;
          break;
        case RIGHT:
          direccion = 4;
```

```
        break;
    } // fin switch
  } // fin if
}
```

Nota: Debe recordarse que la imagen debe estar guardada en la carpeta **data** del sketch.

3.5. Manejo de archivos

A continuación se presentan diferentes ejemplos de cómo se pueden manipular archivos en Processing.

- **Como seleccionar un archivo**

En el siguiente ejemplo, se hace uso del navegador de archivos, para seleccionar en el disco duro un archivo, y al final mostrar en el área de consola, la dirección del archivo seleccionado.

```
String ruta;

void draw() {
}

void mousePressed() {
selectInput("Seleccione el archivo:", "seleccionar");
}

void seleccionar(File archivo) {
  if (archivo == null) {
    println("¡Atención! Ningun archivo fue
```

```
seleccionado");
  } else {
    ruta = archivo.getAbsolutePath();
    println("El archivo selecionado fue: " + ruta );
  }
}
```

Primero se crea la variable **ruta** de tipo **String** en la que se almacena la dirección del archivo que se va a seleccionar. Luego, se debe declarar una función con la cual cargar la dirección del archivo dentro de la variable **ruta** (en este caso se crea la función **seleccionar**). Dentro de la función **mousePressed** se usa la instrucción **SelectImput** que se encarga de llamar a la función **seleccionar** con la cual se abre el navegador de archivos para seleccionar el archivo deseado. Al final de esta función, se muestra en el área de consola la dirección del archivo seleccionado.

Nota: En este ejemplo se debe presionar el mouse para seleccionar el archivo que se desea abrir.

- **Abrir un archivo y mostrarlo en la pantalla**

En el siguiente ejemplo, se hace uso del navegador de archivos, para seleccionar en el disco duro un archivo, y al final mostrar su contenido en la ventana.

```
String ruta;

void setup() {
```

```
background(0);
size(displayWidth, displayHeight);
text("Haga clic para seleccionar un archivo", 10,20);
}

void draw() {
}

void mousePressed() {
background(0);
selectInput("Seleccione el archivo:", "seleccionar");
}

void seleccionar(File archivo) {
  if (archivo == null) {
    println("¡Atención! Ningun archivo fue seleccionado");
  } else {
    ruta = archivo.getAbsolutePath();
    String txt[] = loadStrings(ruta);
    for (int i = 0 ; i < txt.length; i++) text(txt[i], 0, 20*i);
  }
}
```

Primero se crea la variable **ruta** de tipo **String** en la que se almacena la dirección del archivo que se va a seleccionar. Luego, se debe declarar una función con la cual cargar la dirección del archivo dentro de la variable **ruta** (en este caso se crea la función **seleccionar**). Dentro de la función **mousePressed** se usa la instrucción **SelectImput** que se encarga de llamar a la función **seleccionar** con la cual se abre el navegador de archivos para seleccionar el archivo deseado. Al final de esta función, se usa una estructura repetitiva **for** para mostrar en la ventana el contenido

del archivo seleccionado, carácter por carácter, hasta llegar a su final.

- **Guardar texto en un archivo**

En el siguiente ejemplo, se muestra como se puede guardar texto en un archivo mediante la instrucción **saveStrings** con la cual se almacena el texto que ha sido asignado al vector "**lineas**".

```
int lineas = 1;  // Es una Bandera para saber cuántas líneas tiene el archivo
String[] texto = new String[1];
texto [lineas-1] = "Hola como estas?";
lineas++; // se incrementa líneas para aumentar el tamaño del vector
texto = expand(texto, lineas); // agrega una línea más al vector de cadenas
texto [lineas-1] = "bien y tu?";
lineas++; // se incrementa líneas para aumentar el tamaño del vector
texto = expand(texto, lineas); // agrega una línea más al vector de cadenas
texto [lineas-1] = "muy bien, gracias";
saveStrings("hola.txt", texto);
```

Nota: El archivo: **hola.txt** será guardado dentro de la misma carpeta del sketch.

3.6. Llamados desde la línea de comandos

En Processing se puede hacer uso de la línea de comandos para llamar a otros programas o ejecutar comandos del sistema. A continuación, se presentan varios ejemplos.

- **Abrir archivo con el bloc de notas usando la línea de comandos**

En el siguiente ejemplo, se usa la instrucción **open** para hacer un llamado a línea de comandos en **Windows** y abrir un archivo con el bloc de notas el cual debe estar en la carpeta **data**.

```
String comando="temp/archivos.txt";
String param[] = {"notepad"," "};
param1 = comando;
String joined = join(param," ");
open(joined);
```

En GNU/Linux, para hacer un llamado a línea de comandos, el código varía un poco. En el siguiente ejemplo se usa la línea **Process p = exec(command);** para ejecutar el comando, el cual está como tercer parámetro en la cadena **command** (justo después de **"–c"**).

```
String[] command = { "/bin/bash", "-c", "gedit hola.txt" };
Process p = exec(command);
```

En este ejemplo el comando como tal es: "**gedit hola.txt**" que permite abrir el bloc de notas de GNU/Linux y crear un archivo llamado hola.txt, que de ser guardado, quedaría en la misma carpeta del sketch.

PARTE CUATRO:

INFORMACIÓN DE REFERENCIA

4.1. Palabras reservadas del leguaje Processing

Las siguientes listas de palabras reservadas se toman de la API de Processing 2.1.1

Estructura

( ) (paréntesis)
, (coma)
. (punto)
/* */ (comentario multilinea)
/** */ (comentario documento)
// (comentario)
; (punto y coma)
= (asignación)
[ ] (acceso a vectores)
{} (corchetes)
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
popStyle()
private
public
pushStyle()
redraw()
return
setup()
static
super
this
true
try
void

Entorno

cursor()
displayHeight
displayWidth
focused
frameCount
frameRate()
frameRate
height
noCursor()
size()
width

Datos

primitivos

boolean
byte
char
color
double
float
int
long

Compositivos

Array
ArrayList
FloatDict
FloatList
HashMap
IntDict
IntList

JSONArray
JSONObject
Object
String
StringDict
StringList
Table
TableRow
XML

Conversión

binary()
boolean()
byte()
char()
float()
hex()
int()
str()
unbinary()
unhex()

Funciones con cadenas

join()
match()
matchAll()
nf()
nfc()
nfp()
nfs()
split()
splitTokens()
trim()

Funciones con vectores

append()
arrayCopy()
concat()
expand()
reverse()
shorten()
sort()
splice()
subset()

Control

Operadores relacionales

!= (desigualdad)
< (menor que)
<= (menor igual que)
== (igualdad)
> (mayor que)
>= (mayor igual que)

Iteración

for
while

Condicionales

?: (condición)
break
case
continue
default
else
if
switch

Operadores lógicos

! (NO lógico)
&& (Y lógico)
|| (O lógico)

Formas

createShape()
loadShape()
PShape

Primitivas 2D

arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curvas

bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

Primitivas 3D

box()
sphere()
sphereDetail()

Atributos

ellipseMode()
noSmooth()
rectMode()
smooth()
strokeCap()
strokeJoin()
strokeWeight()

Vértices

beginContour()
beginShape()
bezierVertex()
curveVertex()
endContour()
endShape()
quadraticVertex()
vertex()

Carga & mostrar

shape()
shapeMode()

Entrada

Ratón

mouseButton
mouseClicked()
mouseDragged()
mouseMoved()
mousePressed()
mousePressed
mouseReleased()
mouseWheel()
mouseX
mouseY
pmouseX
pmouseY

Teclado

key
keyCode
keyPressed()
keyPressed
keyReleased()
keyTyped()

Archivos

BufferedReader
createInput()
createReader()
loadBytes()
loadJSONArray()
loadJSONObject()
loadStrings()
loadTable()
loadXML()
open()
parseXML()
selectFolder()

selectInput()

Tiempo & fecha

day()
hour()
millis()
minute()
month()
second()
year()

Salida

Área de texto

print()
printArray()
println()

Imagen

save()
saveFrame()

Archivos

beginRaw()
beginRecord()
createOutput()
createWriter()
endRaw()
endRecord()
PrintWriter
saveBytes()
saveJSONArray()
saveJSONObject()
saveStream()
saveStrings()
saveTable()
saveXML()
selectOutput()

Transformar

applyMatrix()
popMatrix()
printMatrix()
pushMatrix()
resetMatrix()
rotate()
rotateX()
rotateY()
rotateZ()
scale()
shearX()
shearY()
translate()

Luces, cámara

Luces

ambientLight()
directionalLight()
lightFalloff()
lights()
lightSpecular()
noLights()
normal()
pointLight()
spotLight()

Cámara

beginCamera()
camera()
endCamera()
frustum()
ortho()
perspective()
printCamera()
printProjection()

Coordenadas

modelX()
modelY()
modelZ()
screenX()
screenY()

screenZ()

Propiedades Material

ambient()
emissive()
shininess()
specular()

Color

Configuración

background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Crear & leer

alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

Imagen

createImage()
PImage

Carga & mostrar

image()
imageMode()
loadImage()
noTint()
requestImage()
tint()

Texturas

texture()
textureMode()
textureWrap()

Pixeles

blend()
copy()
filter()
get()
loadPixels()
pixels[]
set()
updatePixels()

Renderizado

blendMode()
createGraphics()
PGraphics

Sombras

loadShader()
PShader
resetShader()
shader()

Tipografía

PFont

Carga & mostrar

createFont()
loadFont()
text()
textFont()

Atributos

textAlign()

textLeading()
textMode()
textSize()
textWidth()

Medidas

textAscent()
textDescent()

Matemática

PVector

Operadores

% (modulo)
* (multiplicar)
*= (asigna multiplica)
+ (sumar)
++ (incremento)
+= (asignar suma)
- (restar)
-- (decrementar)
-= (asignar resta)
/ (dividir)
/= (asignar división)

Operadores de bits

& (AND bit a bit)
<< (Mueve izquierda)
>> (Mueve derecha)
| (OR bit a bit)

Cálculos

abs()
ceil()
constrain()
dist()
exp()
floor()
lerp()
log()
mag()
map()
max()
min()
norm()
pow()
round()
sq()
sqrt()

Trigonometría

acos()
asin()
atan()
atan2()
cos()
degrees()
radians()
sin()
tan()

Aleatorio

noise()
noiseDetail()
noiseSeed()
random()
randomGaussian()
randomSeed()

Constantes

HALF_PI
PI
QUARTER_PI
TAU
TWO_PI

4.2. Librerías básicas

Las siguientes librerías están incluidas en el entorno Processing 2.1.1.

DXF Export: Permite crear archivos DXF en los que se guarda la geometría, para cargarla en otros programas. Trabaja con gráficos basados en triángulos, incluyendo polígonos, cajas y esferas.

Minim: Es una librería de audio que usa JavaSound para hacerla fácil de usar, además de ofrecer la flexibilidad necesaria para usuarios avanzados.

Network: Permite enviar y recibir datos sobre Internet, mediante clientes y servidores simples.

PDF Export: Permite crear archives PDF de aquello que se grafica con Processing, para escalar e imprimir en alta resolución dichos gráficos.

Serial: Permite enviar datos entre Processing y hardware externo, mediante comunicación serial (RS-232).

Video: Permite leer imágenes desde cámaras, reproducir archivos de video y crear videos.

4.3. Tipos de datos en Processing

Tipo de dato	Descripción
Long	Entero largo
color	Valores de color RGB
double	Decimal largo
char	Caracter
float	Decimal
int	Entero
boolean	Valores boléanos
byte	Valores enteros de 8 bits

- **Tamaño en bits y rangos de valores para distintos tipos de variables**

Tipo	Tamaño	Rango
byte	8 bits	-128 a 127
char	16 bits	-255 a 254
int	32 bits	-2,147,483,648 a 2,147,483,647
float	32 bits	-3.40282347E+38 a 3.40282347E+38
Long	32 bits	-2,147,483,648 a 2,147,483,647
double	64 bits	Mayor que float

4.4. Secuencias de escape

Ciertos caracteres de control, que no son representables, pueden también incluirse en el programa precediéndolos del carácter llamado de escape, que es el

signo \, colocando a continuación el caracter que simboliza la función pertinente. Si el caracter que sigue inmediatamente a la barra invertida no es uno de los especificados aquí, la barra se ignora.

Secuencia escape	Descripción
\n	Salta hacia una nueva línea
\b	Espacio en blanco
\t	Tab
\"	Doble comilla
\'	Comilla simple
\\	Para Backslash
\r	Retorno de carro
\f	Avance de página
\0	Nulo

4.5. Fuentes de referencia

Android. (2014). Android SDK. Consultado el 17 de marzo de 2014 desde https://developer.android.com/sdk/index.html?hl=sk

Arduino. (2014). Arduino development environment. Consultado el 17 de marzo de 2014 desde http://arduino.cc/en/Main/Software#.UydRcdupd8M

Arduino. (2014). Arduino programming language. Consultado el 17 de marzo de 2014 desde http://arduino.cc/en/Reference/HomePage

Cuartas, J.D. (2009). Processing Tricks. Consultado el 17 de marzo de 2014 desde http://sologicolibre.org/projects/cema/index.php?page=Processing+Tricks

Firmata. (2013). Arduino and Processing. Consultado el 17 de marzo de 2014 desde http://playground.arduino.cc/Interfacing/Processing#.Uydawdupd8M

Hiteclab. (2014). Laboratorio Hipermedia de Tecnologías para la Comunicación. [Homepage]. Consultado el 17 de marzo de 2014 desde http://www.hiteclab.co.nr/

Libertadores, L. (2014). Institución Universitaria Los Libertadores. [Homepage]. Consultado el 17 de

marzo de 2014 desde http://www.ulibertadores.edu.co/

Oracle. (2014). JAVA SDK. Consultado el 17 de marzo de 2014 desde http://www.oracle.com/technetwork/java/javase/downloads/index.html

Processing. (2014). Processing. [Homepage]. Consultado el 17 de marzo de 2014 desde http://www.processing.org/

Processing. (2008). Two-Dimensional Arrays. Consultado el 17 de marzo de 2014 desde http://www.processing.org/tutorials/2darray/

Processingjs. (2008).Processing.js . Consultado el 17 de marzo de 2014 desde http://processingjs.org/

Wiring. (2003). Wiring. [Homepage]. Consultado el 17 de marzo de 2014 desde http://wiring.org.co/

El lenguaje de programación Processing fue diseñado específicamente para hacer prototipado rápido de aplicaciones y su principal enfoque es ofrecerle a los artistas, diseñadores y creativos, un entorno de desarrollo amigable y versátil, que no exigiera demasiados conocimientos técnicos, para ser utilizado en la creación de aplicaciones interactivas.

Processing cuenta con una gran variedad de librerías (disponibles de forma libre y gratuita) para que los creadores puedan incorporar nuevas funcionalidades estéticas y técnicas a las aplicaciones interactivas que quieren desarrollar.

Este lenguaje está construido sobre JAVA, lo cual facilita la reutilización de código (Java), para integrarlo dentro de código Processing, sin tener que hacerle modificaciones sustanciales. Esto significa que el código escrito en Processing es multiplataforma, y que se puede ejecutar en diferentes entornos como MAC OS, Windows, GNU/Linux o Android.

En resumen, las principales ventajas por las que se caracteriza Processing tienen que ver con su portabilidad, su comunidad activa, su variedad de librerías disponibles y su buena documentación, así como los foros, accesibles desde la página oficial de Processing.

Este texto pone a disposición de diseñadores, publicistas, comunicadores, creativos y artistas, una guía rápida e intuitiva que les permita aprender a desarrollar rápidamente prototipos de aplicaciones interactivas y comunicativas. Se enfocará en explorar especialmente el lenguaje "Processing", que fue desarrollado inicialmente por Ben Fry y Casey Reas al interior del Media Lab del MIT en el año 2001. Buscando responder al creciente interés por esta temática, el Laboratorio Hipermedia de Tecnologías para la Comunicación (Hitec Lab) adscrito a la Facultad de Ciencias de la Comunicación de la Fundación Universitaria Los Libertadores, desarrolla esta guía introductoria que explora los aspectos básicos del lenguaje "Processing".


ISBN: 978-958-9146-46-0
9 789589 146460